Problem 2: Distributed Auction Service

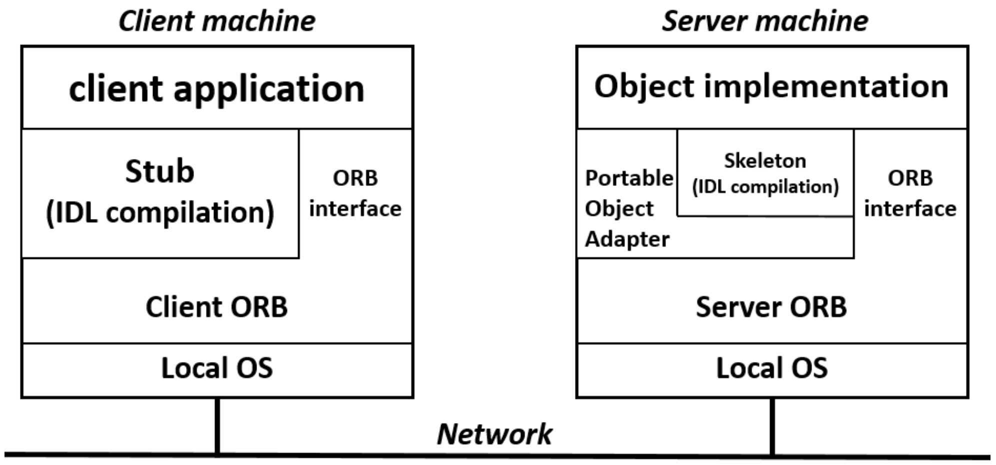
Xu Teng (xuteng@iastate.edu)

Objective

The objective of this project is to use CORBA (Common Object Request Broker Architecture) to build a Distributed Auction Service and Bank Server, which allows the buying and selling of individual items, using an *English* auction protocol (increasing price, current price visible to all parties), and performs basic bank functionalities. This work is completed solely and satisfies the requirements of Phase 1, Phase 2 and Phase 3 (mentioned in instruction documents).

Procedure

The system consists three different types of processes, ***bank server***, ***auction server*** and ***client***, and whole work is programmed in Java language. CORBA is responsible for dealing with communication (i.e., deliver the requests and return the expected values) between client and server, which enables clients to communicate with servers by calling procedures in a similar way to the conventional use of procedure calls in high-level languages.



The CORBA application is developed in the following way:

- **P2.idl:** Interface Definition Language file. Defines application interfaces and servant functions between client and server, by which the object implementation tells clients and servers what operations are available and how to invoke them (i.e., the names of operations, the data types of input arguments and the data types of returned values). IDL file is necessary for creating client stub and server skeleton.

- **P2Server.java:** After using OMG IDL complier to run IDL specs, we will get server-side skeleton. Then compiling them with **P2Server.java** by Java compiler will generate executable server application.
- **P2Client.java:** Similar to server side, by running IDL specs through OMG IDL complier, besides of server-side skeleton, we will as well get client-side stub. Then compiling them with **P2Client.java** by Java compiler will generate executable client application.

Functionalities

- IDL (**P2.idl**): Defines the application interfaces between client and server. To fulfill the requirements of Phase 3, we have three different interfaces:
  - **P2**: Defines *1. The name*, *2. The data types of input parameters*, and *3. The data types of return values* of all the functions between auction server and client.
  - **Bank**: Defines *1. The name*, *2. The data types of input parameters*, and *3. The data types of return values* of all the functions between bank server and client.
  - **P2Callback**: Defines *1. The name*, *2. The data types of input parameters*, and *3. The data types of return values* of all the functions used for informing bidders and sellers about the updates of involved auctions, and used for checking if any user has shut down the application without an explicit logout.
- Server: Consists of the following three classes:
  - **P2Impl**: This servant implements one method for each operation contained by **P2** interface mentioned in **P2.idl**, and is a subclass of **P2POA** (generated by idlj compiler from the **P2.idl**).
  - **BankImpl**: This servant implements one method for each operation contained by **Bank** interface mentioned in **P2.idl**, and is a subclass of **BankPOA** (generated by idlj compiler from the **P2.idl**).
  - **P2Server**: Since Phase 3 requires separate bank and auction services, there are two options in **main()** method (i.e., Bank or Auction) and either of them can be activated individually on different machines as the following manner
    1. Create and initializes an ORB instance
    2. Gets a reference to the root POA and activates the POAManager
    3. Creates a servant instance (**BankImpl** or **P2Impl**) and tells the ORB about it
    4. Gets a CORBA object reference for a naming context in which to register the next CORBA object
    5. Gets the root naming context
    6. Registers the new object in the naming context under the corresponding name (**Bank** or **P2**)
    7. Waits for invocations of the new object from the client
- Client: Client initiates the operations required by Phase 3 and gets the results from either bank server or auction server as the following
    1. Creates and initializes an ORB instance
    2. Obtains a reference to the root naming context

3. Looks up both **Bank** and **P2** in the naming context and receives the references to corresponding CORBA objects
4. Invokes the operations based on the user's input and receives the results from either bank or auction server.

Besides, instead of being a "pure client" (i.e., merely initiates the function call and receives results from server), the client side in this work also acts as a small "callback server", whose interface, **P2Callback**, is defined in **P2.idl**. There are two operations specified in **P2Callback**:

1. *stateChange*: Initiated by server side if there is any update of auctions. Then related users will be invoked and print out the information about the change on its own terminal.
2. *stillAlive*: Initiated by server side and send query to all the users who has logged in but not explicitly logged out yet. If not receiving acknowledgement from any client, that client will be "automatically" logged out on server side.

Measurements

- **Time Complexity**
  - Auction Server: We use HashMap to store the information about users and auctions (denoted as *account_HashMap* and *auction_HashMap* respectively), so the time complexity of either **get** or **put** operation is O(1). And technically, we only walk through the *account_HashMap* or *auction_HashMap* for some operations. Therefore, the total time complexity is O(max{|*account_HashMap*|, |*auction_HashMap*|}).
  - Bank Server: Again, we use HashMap to store the information of each user. And all the operations between users and bank server can be implemented by get/put/remove, so the total time complexity is O(1).

- **Space Complexity**
  As mentioned above, the information is stored in HashMap for both auction server and bank server. Hence, the space complexity of auction server is O(max{# of registered users, # of active auctions}), and of bank server is O(# of registered users).

- **Latency**
  Since CORBA is used to build this distributed system, latency must exist because of communication cost (including network, marshalling arguments/results, and etc.). In total, the time consumption of the whole querying procedure (i.e., client invokes -> server processes -> client gets results), denoted as t_w, and the local processing time (merely the time cost of processing the invoked operations on server side), denoted as t_l, was recorded and repeated for 50 times for different operations. To dismiss the influences of the time complexity of different operations, we will use $(t_w - t_l)/t_w$ to measure that

feature. In sum, the average value is 0.89, and most of values concentrate among 0.97 to 0.99. This result actually meets our expectation. Since the time complexity of the operations on server side is very low (as mentioned above, most of them are in constant time and only couple of them are linear), most of the time consumption will be caused by the communication between client and server.

Steps to Compile and Run

1. Run the IDL-to-Java compiler, idlj, to create stubs and skeletons
   ```
   idlj -fall P2.idl
   ```
2. Compiles the .java files, including the stubs and skeletons generated from Step1
   ```
   javac *.java P2App/*.java
   ```
3. Start orbd and set the port on which the name server to run
   ```
   orbd -ORBInitialPort 1050&
   ```
4. Start Bank Server
   ```
   Java P2Server Bank -ORBInitialPort 1050 -ORBInitialHost 127.0.0.1&
   ```
5. Start Auction Server
   ```
   Java P2Server Auction -ORBInitialPort 1050 -ORBInitialHost 127.0.0.1&
   ```
6. Run the client application
   ```
   java P2Client -ORBInitialPort 1050 -ORBInitialHost 127.0.0.1
   ```
7. Kill the name server and Bank/Auction server